

Metadata-Based Repository for Automated Analysis of Source Data

Description

Technical Field

5 This invention relates to the field of enterprise data management (EDM) applications, and more particularly to the transformation of operational system databases for use in EDM applications.

Background Art

10 Enterprise Data Management (EDM) is the process of sharing data across an organization and making information available to decision makers, regardless of the source of the data. The field has grown rapidly as businesses have discovered the return on investment possible from making decisions based on enterprise-wide data resources. Increasingly, organizations are analyzing
15 current and historical data to identify useful patterns, and to support business strategies. EDM applications include: E-commerce internet portals, customer relationship management (CRM), enterprise resource planning (ERP), business intelligence (BI), and data warehousing.

 Data warehousing is representative of the characteristics of each of these
20 EDM applications, which extract operational data of an enterprise from the enterprise operational data systems and collects it in a multi-dimensional database. The multi-dimensional database supports on-line analytical processing (OLAP) to analyze groups of records that share a common field value, in contrast to on-line transaction processing (OLTP) which accesses
25 individual records in the operational data system relational database. In effect, OLAP extracts business intelligence while OLTP extracts data.

 To permit multi-dimensional analysis the EDM application data must be separated from the operational systems and the required operational system data must be transferred into the EDM application. All of the transferred operational
30 source data must be logically transformed from the operational system model to

the logical and physical structure of the data warehouse, which aligns with the business structure. As an example, a data warehouse may combine data from different source applications such as sales, marketing, finance, and production, giving it the ability to cross-reference data from these applications. This

5 transfer and transformation of the data between system models is referred to as “data migration”.

Assuming that the data model, i.e. the logic and structure, of the particular EDM application is known, the major step in data migration is the transformation of the source data, which is necessary to make it independent of
10 the relational data model of the operational systems as well as have it fit into the EDM data model. Attributes of the source data that are essential to the operational system may be unnecessary for the particular EDM application, and are not loaded into the warehouse. This step typically involves “data cleansing” or “data staging” processes which are labor intensive and tedious in a data
15 warehousing project. This transformation involves the major steps of reverse engineering the source data, its form, and its database structures to determine its metadata, adapting the source data to fit the data structure of the EDM application, and loading the transformed source data to the EDM target database.

20 In varying degrees, this requires the EDM application developer to: (i) analyze the source databases to identify the type and form of the source data and the structure and operational system of the source databases, to determine their relevance to the EDM target database; (ii) load the source data from the legacy databases to an intermediate database using extract, transform, and load (ETL)
25 software tools or hand-coded programs; (iii) massage, cleanse and manipulate (“transform”) the source data in the intermediate database into the form needed for the EDM application; and (iv) load the transformed source data into the EDM application, and format it in those schemas necessary for OLAP (on-line application processing), or other EDM applications.

In the prior art these steps are performed manually, by a team of developers, including those of ordinary skill in the art of software programming. While there are now ETL software tools that automate the process of extracting and transforming data from the source database to the EDM application, these
5 ETL tools recognize and rely on the source metadata (the source data structure) available from the source database management system (DBMS) to generate the EDM application code. However, the program developer can only determine the source metadata through the source data DBMS documentation (if available), interviews with the customer (if they have the knowledge), and a
10 lengthy and laborious trial and error process.

The assumption by a developer that the required metadata is available is one of the weaknesses of the prior art methods; other weaknesses include the manual labor involved, as opposed to automated reverse engineering. Source documentation may not be available or, if available, it may be inaccurate. This
15 results in the need for experimentation, which is a manual process with associated cost and delay and risk of error. Even with experimentation there may not be an opportunity to confirm the accuracy of the results because the personnel needed for interpretation of the data may have incomplete information or may have left the company. As a result there is a degree of guesswork and a
20 process where the migration of source data into the warehouse is debugged downstream in the development cycle. This may translate into a significant financial loss for the enterprise and even the failure of the project. As long as the data migration process consists of these multiple, labor-intensive independent steps, the failure rate and cost for these projects will remain high.

25

Disclosure of Invention

One object of the present invention is to provide method and apparatus for acquiring the metadata of a source database in a manner which allows the acquired metadata to be used to generate diverse type EDM applications. A
30 further object of the present invention is to create an architecture for a data

repository which defines the metadata characteristics of a generic model data source which may be commonly used to generate a large number of different EDM applications. A still further object of the present invention is to provide method and apparatus for acquiring the metadata of a data source with minimum error. A yet still further object of the present invention is to provide method and apparatus for acquiring the metadata of a data source with minimum data redundancy.

According to one aspect of the present invention, a method of obtaining the metadata of a data source, comprises the steps of creating a data repository having an entity structure which defines the metadata characteristics of a generic model data source, accessing the data source to determine its construct, configuring the data repository entities to reflect the construct of the data source, and analyzing the data source in response to the configured data repository entities to obtain the source metadata. In further accord with this aspect of the invention, the entity structure of the data repository is independent of the construct of the data source. In still further accord with this aspect of the invention, the entity structure of the data repository includes plural entities, each entity corresponding to a different aspect of a generic model data source, and each entity having atomic elements which define the metadata attributes of a generic entity, whereby the plural entities collectively define the metadata characteristics of a generic model data source.

According to another aspect of the present invention, the step of analyzing the data source to determine its metadata includes the steps of obtaining those elements of source data which correspond to the metadata attributes of the configured data repository entities, and recording the obtained elements of source data in the data repository, each in association with their corresponding metadata attribute. In further accord with this aspect of the invention the step of analyzing further includes the steps of inferring selected aspects of the data source structure on the basis of the recorded elements of source data, and recording those inferred data source aspects in the data

repository for review by an operator; who may either accept or modify one or all of the inferences.

The present invention automates the process of building an EDM system application, and in its best mode the method is embodied as a software application. It employs an exhaustive reverse engineering process to analyze the source data and acquire a comprehensive set of source metadata. The invention then uses this information to build a model of the data. As part of the analysis process the invention provides recommendations for the target EDM application database, such as primary keys, foreign keys, table splits, normalization, dimensions, measures, and views of the data. The result is an optimal target database for the EDM application.

The present invention allows users to migrate multiple disparate systems by providing a complete understanding of the metadata and generating the ETL programs to merge the data. Since the code is automatically generated, the speed of implementation is dramatically increased. Because the metadata is based on the actual source data, it is of the highest degree of accuracy.

These and other objects, features, and advantages of the present invention will become more apparent in light of the following detailed description of a best mode embodiment thereof, as illustrated in the accompanying Drawing.

Brief Description of Drawing

Figure 1 is a figurative system diagram of one embodiment in which the present invention may be used;

Figure 2 is a schematic block diagram of one element of the embodiment of Figure 1;

Figure 3 is a flow chart diagram illustrating the functions performed by the invention in the embodiment of Figure 1;

Figure 4 is a flow chart diagram illustrating further details of one of the functions illustrated in Figure 3;

Figure 5 is an entity relationship diagram illustrating the structure and functional relationships of the data entities established within the data repository of the present invention;

Figure 6 is an entity relationship diagram used in the description of the functional relationships between the entities of Figure 5;

Figure 7 is an entity relationship diagram used in the description of the functional relationships between the entities of Figure 5 in performing the data analysis method of the present invention;

Figure 8 is another entity relationship diagram used in the description of the functional relationships between the entities of Figure 5 in performing the data analysis method of the present invention;

Figure 9 is another entity relationship diagram used in the description of the functional relationships between the entities of Figure 5 in performing the data analysis method of the present invention;

Figure 10 is another entity relationship diagram used in the description of the functional relationships between the entities of Figure 5 in performing the data analysis method of the present invention;

Figure 11 is another entity relationship diagram used in the description of the functional relationships between the entities of Figure 5 in performing the data analysis method of the present invention;

Figures 12A and 12B, together, are another entity relationship diagram used in the description of the functional relationships between the entities of Figure 5 in performing the data analysis method of the present invention;

Figure 13 is another entity relationship diagram used in the description of the functional relationships between the entities of the data repository in the generation of a migration specification; and

Figure 14 is another entity relationship diagram used in the description of the functional relationships between the entities of the data repository in the generation of an OLAP cube EDM application.

Best Mode for Carrying out the Invention

The present invention is to a method and apparatus for analyzing the relations, attributes, and rows or records of data stored within a source database to determine its metadata. It analyzes the source database in an elemental
5 sequence that is defined by the logical and structural model of a data repository having defined entity relationships, so as to provide a comprehensive metadata model of the source data. The detailed information provided by this metadata model is sufficient to provide a generic description of the source data that can be used to generate program code for a plurality of EDM applications with
10 different database models and operating systems.

As defined in detail hereinafter, a central feature of the present invention is the ordered sequence and elemental detail of the analysis performed on the source data. This analytic discipline is governed by demand of the data repository model, which functions as a checklist for the metadata details
15 required. This structured approach ensures both a comprehensive analysis and a consistency between the analysis performed on each source database, to provide the highest degree of accuracy in results. While the same comprehensive analysis, consistency, and accuracy can be achieved by using this structured approach manually, in its best mode the invention is embodied in software. The
20 software embodiment may be provided in any of a variety of known program languages for use in any of a variety of applications, using any of a number of known signal processing devices and operating system protocols.

Figure 1 is a schematic illustration of a best mode embodiment 20 of the invention in a network-based, client/server application, for use in analyzing the
25 data stored in a source database 22. The network may be a local or wide area network with connectivity to the necessary file and database servers. However, it should be understood that the Figure 1 embodiment is only one of several alternative embodiments that may be used to provide connectivity. The invention is not limited to use in a network and, as known to those skilled in the
30 art, any of a number of alternative methods and configurations may be used to

connect the client/server to the source database 22. Similarly, the analyzed source database 22 may have any type data construct, with any type source database management system (DBMS) software to manage data access. For ease of reference, the source database and associated DBMS are here designated
5 as the source DBMS 22.

As described previously, in its software embodiment the invention performs the automated analysis of the characteristics of the source DBMS 22 to determine its construct, form, and operating system characteristics to a degree which permits the invention to make inferences regarding the characteristics of
10 individual data entries as well as their column and table construct within the source database. This analysis is performed in a sequence of defined steps; each step being associated with a different phase of the analysis construct. At the conclusion of each phase the invention makes inferences regarding the source data construct. The inferences are dependent on the analysis results, and are
15 stored with the analysis results within the data repository 37. Succeeding phases are also, to a degree, dependent on the inferences made in the preceding phase. In the best mode embodiment, the invention provides the human operator, such as the EDM application developer, the opportunity to review the inferences made and either accept or modify them before proceeding to the next
20 phase.

In the best mode embodiment, the human operator is given access to the method through a graphical user interface (GUI) 24, comprising a known type signal processor 26, a GUI display 28, and an operator pointer/command entry device 30 (e.g. keyboard, mouse, etc.). Depending on the size of a given EDM
25 project, there may be several GUI interfaces for team use. The GUI 24 is connected through its processor's communication port, and the network 32, to one or more application servers, such as the analysis server 34. Since the operational steps of the present method are machine time dependent, the analysis may be performed on multiple analysis servers if desired. However,
30 only one analysis server is shown in Figure 1 for simplicity of illustration.

The analysis servers 34 communicate with the GUI 24 through a messaging server 36, which functions as a message handler in facilitating communications exchange between all components of the method's embodiment. In the illustrated embodiment, the analysis servers 34 perform analysis of the source DBMS 22 and store the results of each analysis in a data repository 37, which is a known type, non-volatile signal storage medium. The analysis results stored in the data repository 37 are then displayed to the operator through the GUI 24.

The servers 34, 36 each include a known type signal processor 38, 40 and optional viewable display 42, 44. The signal processors 26, 38 and 40 for the GUI 24, the analysis server 34, and the message server 36, are each of a known type, which embody a central processing unit (CPU) architecture of the type shown in simplified form in Figure 2 by the signal processor 46. Referring to Figure 2, the signal processor 46 includes one or more CPUs 48 connected through a universal signal bus (USB) 50 to a plurality of input/output (I/O) devices 52-55, and to signal storage devices, such as one or more non-volatile read-only memory (ROM) devices 58, and one or more volatile random access signal memory (RAM) devices 60. The I/O devices 52-55, respectively, connect the USB 50 to the signal processor's printer port 62, the operator input device 64 (e.g., keyboard, mouse), the viewable display 66, and the processor's communications port 68.

Although the data repository 37 is shown as a separate signal storage device in the best mode embodiment of Figure 1, it should be understood that the data repository may alternatively reside in the signal memories resident in any one or several of the signal processors which are used to execute the method. In other words, the user may elect the location of the data repository as deemed suitable for a given application. Similarly, as may be more easily understood following the detailed description below, the structure of the data repository may be implemented in several segments, each in different storage devices, as deemed suitable by those skilled in the art.

In the best mode embodiment the GUI 24 and analysis server 34 operate in a client-server protocol. Different files of the software program embodying the invention are loaded into the memories of both. The files associated with the interactive display are installed in the GUI signal processor 26 and those
5 associated with the data analysis functions are installed in the signal processor 38 of the analysis server 34. The files installed in the analysis server include an application program interface (API) to provide for server access to the source DBMS 22. The API may of any known type. In a best mode embodiment the software includes an Open Database Connectivity (ODBC) application
10 programming interface developed by Microsoft® Corporation.¹

As known, the ODBC application defines a low-level set of calls which allow the client application running on the analysis servers 34 to exchange instructions and share data with the source DBMS 22, without the two having to know anything about each other. It does this by inserting a “database driver”
15 middle layer between the application and the DBMS software. The driver translates the application data queries into commands that the DBMS understands.

Prior to the actual analysis of the source DBMS the developer/operator will obtain access (logon) information for the source DBMS 22, such as an
20 assigned user ID and password which allows the operator to log into the source DBMS 22 from the GUI. The analysis server 34 initiates contact with the DBMS of the source DBMS 22 on command of the operator from the GUI 24. The server 34 communicates with the source DBMS 22 using a structured query language (SQL) format and the ODBC library of function calls. The analysis
25 server uses the “catalog” and “retrieval” function calls to have the source DBMS 22 download the available source metadata.

This is the first of several process steps performed by the analysis server in the overall method 70, as shown in the flow chart of Figure 3. Referring to Figure 3, the server enters the method 70 at the ENTER 72, and performs the

¹ Microsoft is a registered trademark of the Microsoft Corporation.

process step 74 of accessing the source DBMS following logon to the source DBMS by the operator. Once connected, the server performs process step 76 to query the source DBMS, using SQL commands and the ODBC library of function calls, to obtain all of the available source metadata. As described in
5 detail hereinafter, the elements of the downloaded metadata are stored in respective attribute areas of the structured entity data repository 37.

With the downloaded metadata recorded in the data repository, the analysis server then performs the process step 78 to analyze the source database in the elemental manner referred to above. The analysis results are stored in a
10 separate area of the repository and compared with the metadata entered in the repository in the download process 76. In doing this it verifies consistent values or rejects conflicting metadata elements entered in the earlier download. It is the metadata obtained with this analysis step that is principal, since it is obtained directly through analysis of the data itself and must, therefore, be
15 accurate. Alternatively, the metadata that is resident in the source is that entered by the source DB developer, based on their belief in its accuracy. In the ideal situation the two sets of metadata are in agreement and corroborate each other, which is the best case. At the end of step the analysis exits the method at 80.

20 Figure 4 is a further detailed flowchart of the data analysis process 78 of Figure 3. However, before describing the details of the analysis process it is first necessary to describe the logical and structural model of the data repository 37, which the present inventive method uses to store the source metadata to be migrated to an EDM application, and which facilitates the ordered approach
25 executed by the analysis process. The architecture of the repository is “metadata-centric”, in that the atomic elements of the repository data storage structure themselves define the metadata characteristics to be obtained for the source data that is being modeled. Its data storage structure is independent of the structure, definitions, or constructs of the source data structure, so as to

provide a generic storage model that is suitable as a data source from which EDM applications may be generated in any desired model, format, or language.

Figure 5 is a summary entity-relationship diagram for the data repository 82, which contains the data entities 84-92 that represent the data structure of a generic source database (i.e. a database having one or more tables, each table having one or more columns, and each column having one or more rows), as well as the key fields ("keys") of the database tables and the relationships between the tables. The entities are described below. They each include in their name the abbreviation "Dict" for "Dictionary". This term is referential, but is used here to connote the definitive nature of the analysis performed by the present invention in determining the source metadata.

The data repository entities are:

DictDb

The DictDB entity 84 is structured to hold all of the "standard" informational properties of a generic source database. At the minimum this entity includes the physical and logical name of the source database; the type/category (e.g. RDBMS, SEQUENTIAL, etc.) of the source database; the physical location of the source database; and the information required to establish a connection to the source database.

DictTable

The DictTable entity 85 is structured to hold all "standard" informational properties of a generic table within a generic database. At the minimum this entity includes the physical and logical name of the table; the type of table (e.g. CSV, FIXED, DELIMITED, RELATIONAL, ODBC, SEQUENTIAL, etc.); the physical location of the table; and the information required to establish a connection to the table.

DictColumn

The DictColumn entity 86 is structured to hold all "standard" informational properties of a generic column within a given generic table and generic database. At the minimum this includes the physical and logical name of the

column; the physical sequence (placement) of the column within the table; and the data type characteristics of the column data.

DictDbPropertyList

The DictDbPropertyList entity 87 is a collection of additional database property types that may optionally be applied to the definition of the database. This collection supplements the "standard" database properties which are attributes of the DictDb entity 84. It allows for the expansion of the DictDB attributes without restructuring the Dict DB entity itself. Every instance of this entity represents a single optional property that may be realized by creation of an instance of the DictDbProperty entity 88 (described below) for a given database. This entity is simply a list showing the valid optional properties, such that one or more may be utilized for a given database.

DictDbProperty

The DictDbProperty entity 88 is a collection of additional informational properties for a given database, each of which is of a type defined by an instance of the DictDbPropertyList entity 87.

DictTablePropertyList

The DictTablePropertyList entity 89 is a collection of additional table property types that may optionally be applied to the definition of a given table. This collection supplements the "standard" table properties which are attributes of the DictTable entity 85. Every instance of this entity represents a single optional property that may be realized by creation of an instance of the DictTableProperty entity 90 (described below) for a given table. This entity is simply a list showing the valid optional properties, such that one or more may be utilized for a given table.

DictTableProperty

The DictTableProperty entity 90 is a collection of additional informational properties for a given database, each of which is of a type defined by an instance of the DictTablePropertyList entity 89.

DictColumnPropertyList

The DictColumnPropertyList entity 91 is a collection of additional column property types that may optionally be applied to the definition of a given column. This collection supplements the "standard" column properties which are attributes of the DictColumn entity 86. Every instance of this entity represents a single optional property that may be realized by creation of an instance of the DictColumnProperty entity 92 (described below) for a given column. This entity is simply a list showing the valid optional properties, such that one or more may be utilized for a given column.

DictColumnProperty

The DictColumnProperty entity 92 is a collection of additional informational properties for a given column, each of which is of a type defined by an instance of the DictColumnPropertyList entity.

In practice, each of the above described entities 84-92 may be implemented as a separate table within the data repository 37 (Figure 1), i.e. the physical memory storage device which is the underlying structure of the repository. The relationships between the entities are generally "one-to-many," which means that an instance (or "attribute") of one entity (e.g. the parent) may be related to many instances (attributes) of the another entity (e.g. the child).

In Figure 5, these relationships are evidenced by "relationship lines" 93 between the entities 84-92. A solid color circle terminating a relationship line 93 at an entity identifies the terminal entity as the child in the relationship. To assist the reader in understanding the nature of the relationship, each line also includes a verb phrase in italics to suggest the meaning of the relationship. As an example, the phrase "contains" adjacent the relationship line 93 between the DictDb entity 84 and the DictTable entity 85 identifies the fact that the DictTable entity is included within the DictDB entity. The terminating solid circle at the terminal end of the relationship line also identifies the DictTable entity as the child of the parent DictDb entity.

As stated above, the repository of the present invention provides representation of the data structures of a generic model database. The generic attributes of the data objects of the model database are represented as properties of the DictDb, DictTable and DictColumn entities. In the summary Entity-Relationship Diagram of Figure 5 the entities 84-92 are only shown with their respective "primary keys" 94-102, which are the attribute characteristics that uniquely identify occurrences of each entity. For example, the "(FK)" designation following the attribute "DbName" 95 in the DictTable entity 85 indicates that this attribute supports a foreign key used in the DictTable entity relationship to the DictColumn entity 86.

The principal attributes 104-112 of the Figure 5 entities 84-92 are listed in Appendix I. Referring to Appendix I, the identified attributes for each entity are those considered suitable to define a generic form of the entity that they are associated with. Collectively, they define all of the attributes foreseen as necessary to permit the data modeling of any known type of source database, whether it is a relational database or a collection of data files. It does this through its ability to model all of the characteristics of the source DBMS data, including:

- (i) the data type characteristics of individual source data elements;
- (ii) the grouping of elementary data items into a collection, such as a file or table;
- (iii) the grouping of these collections into an aggregate database;
- (iv) the relationships between data elements within a given collection; and
- (v) the relationships between the collections themselves.

The presence of specialized attributes of source data objects, if they are required to fully describe an instance of that object, are represented using the <object-type>PropertyList entities 87, 89, and 91, and the <object-type>Property entities 88, 90, and 92. The instances of the DictDbPropertyList entity 87, for example, define the valid specialized attributes that may be defined for any DictDb instance. The PropertyName (FK) (foreign key relationship) referencing

DictDbPropertyList 87 from DictDbProperty 88 ensures that only valid specialized properties may be used to further define a DictDb entity 84 instance.

The data repository model also provides for dependencies, or relationships, between tables in a source data structure. These relationships are modeled as shown in the summary Entity-Relationship diagram of Figure 6. The entities illustrated in Figure 6 which were previously described and illustrated in Figure 5 (DictTable 85 and DictColumn 86), are shown with their prior reference numerals, while the added entities in Figure 6 include:

DictKey

The DictKey entity 114 holds all the information for a given key of a table. Several types of keys may be represented, including primary keys, foreign keys, and alternate keys. The DictKey entity 114 can be used to represent both the logical structure and physical characteristics of the table keys.

DictKeyColumnMap

The DictKeyColumnMap entity 116 lists the columns of a table that are used to construct a given key of that table. This entity includes the appropriate sequencing of those columns as required to implement the key.

DictRelation

The DictRelation entity 118 holds the definition of a relationship between two tables, representing one table as the "parent" and one table as the "child" in the relationship. A single table may serve as both parent and child. A given instance of DictRelation 118 is associated with the parent and child tables. A given instance of DictRelation 118 is also associated with the instance of DictKey 114 which represents the primary key of the parent table, as well as the instance of DictKey 114 which represents the foreign key of the child table.

A given relationship (e.g. an instance of a DictRelation entity 118) is represented as having exactly one parent and one child, to indicate the customary roles served by, for example, two related tables in a relational database. A single table may serve as both parent and child. Customarily, the parent table will also include the definition of a primary key (i.e. an instance of

a DictKey entity 114), which serves to uniquely identify every row in the table. To support the relationship from the child to the parent, the child table may include the definition of a foreign key (another DictKey instance), which corresponds to, or maps to, the primary key of the parent.

5 The columns that include a key of any type are represented using the DictKeyColumnMap entity 116, which relates the key to a set of one or more DictColumn occurrences, each of which is defined for the containing DictTable instance. The full list of attributes for each entity shown in Figure 6 may be found in Appendix I.

10 The generic model data repository described above is capable of representing any collection of database-like data without the need to alter its own data structure or altering the software used to access the data from the repository. In the event that it is necessary to add distinct properties of a given source database in the repository, the use of the several "PropertyList" and
15 "Property" entities allow these properties to be added without modifying the DictDB, DictTable and DictColumn entities 84-86. The completeness of the metadata capture provided by the repository also ensures that software which is based on the repository structure would be able to connect to any type source database without requiring additional information.

20 It should also be emphasized, that the generic nature and the comprehensiveness of the model ensures that all details of a source database structure can be obtained to such a complete degree that the source database itself can be replicated from the repository metadata. The repository metadata is not specific to any particular database technology or vendor. As a result, it is
25 capable of being used to generate a wide variety of EDM application applications, in a multiple of different presentation materials and different technologies.

 Having described the entity structure, the analysis server 34 performs the data analysis routine (78, Figure 3) of the source DBMS 22 (Figure 1) in several
30 phases. As shown in the flow diagram of Figure 4, the server 34 enters the

process 78 at 120, and performs an initialization step 122. This is an initial step in which the source database definition is prepared for data analysis. This preparation involves the collection of the database table and column definitions into a parallel area of the data repository 37 (Figure 1). It also establishes the configuration settings that govern the data analysis process so that the data repository can govern and track the procedures that take place in the analysis steps.

To further explain the initialization process 122, and as described in further detail with respect to Figure 7, there are three data repository entities that comprise the analysis phase. They include a DbAnalysis entity, a TableAnalysis entity, and a ColumnAnalysisConfig entity. Many of the attributes of these entities are utilized as configuration settings that govern one or more phases of data analysis. All such attributes are configurable by the user, which grants considerable control over the algorithms to the user.

DbAnalysis

Some of the configuration attributes of the DbAnalysis entity can be overridden, if desired, at either the table level (for a given table) or the column (for a given column). In these cases, the corresponding set of attributes of the TableAnalysis entity or the ColumnAnalysisConfig entity are utilized.

Column Analysis

During the first step of the source data analysis, which is Column Analysis, a data sample is obtained for every analyzed column. The method by which the sample is determined is governed by the SamplingMethod, SampleParameter and SampleMaxRows attributes as described in Table A below:

Table A

SamplingMethod	SampleParameter	SampleMaxRows
N-th	value of N	value for max number (nbr) of rows
Percentage	value for %	percentage for max nbr of rows
Random Set	value for N	N/A

Method "N-th" involves selecting the N-th data record up to a maximum number of rows (SampleMaxRows).

Method "Percentage" involves getting a random number/percent between 0 and 100% for each row read. If random percent is less than or equal to
5 SampleParameter, then this row is included in the set of sample data, up to the specified maximum number of rows (SampleMaxRows).

Method "Random Set" involves calculating a series of random row numbers up to the total number of rows in the set of input data. Inclusion in the set of sample data occurs if the N-th row number (during sequential processing)
10 is also present in the randomly-generated set of row numbers. SampleMaxRows does not apply since SampleParameter is in effect the exact size.

The following six "threshold" attributes collectively support enhanced inference capabilities:

(i) FunctionalDepThreshold - During Table Analysis, the minimum
15 percentage of values in a given column which must be dependent on the determinant column(s) before the dependency is recognized and recorded.

(ii) CommonDomainThreshold - During Cross Table Analysis, the minimum percentage of the values of a column that must participate in a domain intersection before a common domain situation will be identified
20 and recorded using the CommonDomain entity. The percentage of BOTH candidate columns must meet or exceed this threshold.

(iii) NullityThreshold - During Column Analysis, if the percentage of null/empty values for a given column exceeds this threshold, then the associated inference (that the column permits null/empty values) will be
25 made. This process enables the identification of column nullity even when a degree of "dirty data" is present.

(iv) ConstantThreshold - During Column Analysis, if the percentage of inferred constants for a given column exceeds this threshold, then the associated inference (that the column represents a constant value) will be

made. This process enables the identification of constants even when a degree of “dirty data” is present.

(v) UniquenessThreshold - During Column Analysis, if the percentage of unique, non-null values for a given column exceeds this threshold, then the associated inference (that the column represents unique, non-null values) will be made. This process enables the identification of column uniqueness even when a degree of “dirty data” is present.

(vi) AllDistinctValuesTreshold - During Column Analysis, if the percentage of non-null values which are also unique exceeds this threshold, then the associated inference (that the column represents either non-null or unique values) will be made. This process enables the identification of columns with unique values even when a degree of “dirty data” is present.

Dealing with Case Sensitivity

During Column Analysis, character data values must be processed correctly regarding the case of the characters. The CaseSensitivity and CaseChange attributes are used to determine the case handling. If the CaseSensitivity is set to ‘Y’, then the data values are processed unchanged (their case is preserved during processing). If set to ‘N’, the CaseChange attribute specifies the desired case. This is part of the process of building canonical data values for later processing.

CaseChange – If set to 'Upper', character data values are converted to uppercase. If set to 'Lower', character data values are converted to lowercase. If CaseSensitivity is set to ‘Y’, then this attribute is ignored.

TreatAsBlobSize

During Column Analysis, if the length of a data value is greater than or equal to this setting, then a checksum value is generated to represent this data value instead of processing and storing the actual large data value. This is part of the process of building canonical data values for later processing.

NullStringValue

During Column Analysis, the specified value is used as a NULL value for

character string data types.

NullNumericValue

During Column Analysis, the specified value is used as a NULL value for numeric data types.

5 ColumnAnalysisConfig

All attributes of the ColumnAnalysisConfig entity exist only as column-specific overrides for corresponding attributes (settings) which may also be specified at the table and/or database level.

TableAnalysis

10 During Table Analysis the DetColumnCountUsed entity represents the maximum number of determinant columns that were considered during the last run of Table Analysis for this table. If Table Analysis is subsequently repeated, this knowledge can be used to optionally skip a given table if the currently-selected maximum number of determinant columns has not changed. This is a
15 performance optimization that is especially relevant since Table Analysis has the potential to be a long-running process. All other attributes of the TableAnalysis entity exist only as table-specific overrides for corresponding attributes (settings) which may also be specified at the database level.

Determining Column Key Coverage

20 During Primary Key Analysis, an aggregate dependency is only recognized as a candidate primary key if the percentage of table columns that are dependent upon the determinant column(s) exceeds the KeyCoverageThreshold. Assume:

25 D = the number of determinant columns associated with the aggregate dependency;

D = the number of dependent columns associated with the aggregate dependency; and

T = the total number of table columns that are NOT constants

The percentage for a given column is calculated as: $d / (T - D) \times 100$

Cross Table Analysis

During Cross Table Analysis, the `SynonymRequiresTypeMatch` setting is used to specify whether or not two columns being considered for common domains are required to share the same basic data type. The default is 'Y'. If set to 'Y', then `ColumnInfoInferred`. `DataTypeChosen` must be identical for both columns being considered.

The summary entity relationship diagram of Figure 7 shows the data repository entities that are used to prepare for and control the data analysis process. Referring to Figure 7, as earlier described and illustrated with respect to Figure 5, the data repository structure includes an “instance” of a `DictDb` entity 84 for each source database to be analyzed, and the initialization step 122 provides an instance of a `DbAnalysis` entity 124. The `DbAnalysis` entity 124 includes the `DbName` primary key 126, as well as configuration settings and other information shown in Appendix I. Similarly, the initialization step 122 creates an instance of a `TableAnalysis` entity 128, with its primary keys 129 and its Appendix I settings, for every instance of a `DictTable` entity 85 within the source database being analyzed. Finally, a `ColumnAnalysisConfig` entity 130, with primary keys 131 and Appendix I settings is created for every instance of a `DictColumn` entity 86 within the `DictTable` entity 85.

As may be evident, `DbAnalysis` 124, `TableAnalysis` 128, and `ColumnAnalysisConfig` 130 are each structural entities of the data repository, which are used, as necessary for a given source database configuration, to govern the incremental steps of the analysis through their individual configuration settings (see Appendix I). As seen from the relationship lines 132-134 the individual analysis entities 126, 128 and 130 are each the child of the parent `DictDB`, `DictTable` and `DictColumn` entities 84-86.

Referring again to Figure 4, following the initialization step 122 the process 78 next performs a column analysis step procedure 136. The column analysis is the process of examining the values of individual data values in each column of the source database, and recording what can be learned and/or

inferred about them. During the column analysis procedure each column in each table of the source database is examined individually. Many characteristics of the data in a column are recorded, including:

- (i) the minimum, maximum, and average length of values, both before and after any preprocessing (such as removing leading spaces or zeros);
- (ii) the minimum and maximum precision (to the left of the decimal point) and scale (to the right of the decimal point) for numeric values;
- (iii) the minimum and maximum numeric values;
- (iv) the number of empty values, number of NULL values, and number of non-NULL non-empty values.;
- (v) the number of distinct values;
- (vi) the basic data types encountered: TinyInt (1 - 2 digits), SmallInt (1 - 4 digits), Integer (5 - 9 digits), Decimal, Float1, Float2, String, and DateTime; and
- (vii) for DateTime values, the specific date/time formats found (referred to as “ExtendedType”).

Also, the least frequent and most frequent values are captured so that they can be reviewed if needed. These are the actual source data values, without regard to data type (character or numeric). They represent partial distributions of the source data. They are made available for review by the operator as supporting data for the inferences made by the method.

In addition to recording the observed source column data characteristics, the column analysis procedure 136 also makes certain inferences about the data, such as:

- (i) the data type, its precision and its scale;
- (ii) the date/time format for DateTime data types;
- (iii) whether or not NULLs are permitted;
- (iv) whether or not the column contains a constant value; and
- (v) whether or not the column values are unique.

Many of the functions performed during the column analysis procedure are subject to the configuration settings. These settings are stored in the

DbAnalysis entity (for settings global to a database), the TableAnalysis entity, and the ColumnAnalysisConfig entity.

The repository entities used during column analysis are shown in the entity summary diagram of Figure 8. As shown in Figure 8 the

5 ColumnInfoInferred entity 138 (with primary keys 139) and ColumnAnalysisResults entity 140 (with primary keys 141) are both children of the ColumnAnalysisConfig entity 130. Each instance of these entities 138, 140 represent the inferences and results of analysis of one column of a source database column. The ColumnAnalysisResults entity 140, together with a
10 ColumnLeastFrequent entity 142(with primary keys 143), a ColumnMostFrequent entity 144(with primary keys 145), and a ColumnDataType entity 146(with primary keys 147) represent raw results of the analysis process.

The DateFormat entity 148 (with primary keys 149) provides the
15 definitive list of all recognizable date/time formats, from which a candidate date format may be chosen for a given ColumnDataType instance. The ColumnInfoInferred entity 138 captures the recommended characteristics for each analyzed source data element. This entity forms the basis for generation of data models from an analyzed database. Appendix I shows all the attributes of
20 the entities involved in Column Analysis.

Referring again to Figure 4, following completion of the column analysis procedure 136 the process pauses at step 150 to provide a review phase in which the operator may review the inferences made by the present method in its column analysis phase. The column analysis inferences are a necessary
25 antecedent to each of the analysis phases to follow. In this review phase 150 the operator may either accept or modify the inferences made by the invention in the column analysis phase.

Following review the operator commands the table analysis procedure 152. The table analysis process 152 is one of examining the relationships
30 between all of the columns of a source database table, and identifying all of the

functional dependencies between them. Such a functional dependency exists within a given table T if a column C1 is dependent on, or determined by one or more other columns (e.g. a set "S" of columns) within the table. S is a set of one or more columns that does not include the dependent column C1. The set of columns S on which C1 is dependent is known as the determinant. Therefore, a dependency exists if, for a given value of the determinant S, the values of the dependent column C1 are always the same.

Identifying these functional dependencies is important because it supports the identification of primary keys and relationships, and supports the process of table normalization. The data repository entities used in the table analysis procedure are shown in the summary entity diagram of Figure 9. Many of the functions performed during the table analysis process are subject to configuration settings. Those of the settings which are global to a source database are stored in the DbAnalysis entity (124, Figure 7) and the settings which are specific to a table selected for analysis are stored in the Table Analysis entity (128, Figure 7).

Figure 9 illustrates the working relationship of the TableAnalysis 128 with those data repository entities which are involved with the table analysis procedure. The Aggregate Dependency entity 154 (with primary key 155) represents those functional dependencies detected within a table being analyzed. It is named "aggregate" because it addresses all columns that are dependent upon a given determinant, as opposed to the definition of a functional dependency, which addresses only a single dependent column. Collecting all such dependent columns into a single aggregate dependency simplifies the process of identifying and processing dependencies.

For a given Aggregate Dependency entity 154, the set of columns that comprise the determinant is recorded using the Determinant Column entity 156 (with primary keys 157). One occurrence of the Determinant Column entity will exist for each such determinant column found in the analysis. The Dependent Column entity 158 (with primary keys 159) is used to represent each

dependent column for a given dependency. Appendix I lists all the attributes of the entities involved in the table analysis process.

Referring again to Figure 4, following completion of the table analysis procedure 152 the present method again pauses at 160 to provide a review phase
5 in which the operator may review the inferences made in the table analysis. Once again the operator may either accept the table analysis inferences or modify them before proceeding. Following review the operator commands the Primary key analysis 162.

The primary key analysis is the process of identifying all candidate keys
10 for each analyzed source database table. A primary key is a set of columns (i.e. the determinant of a dependency) upon which all remaining columns are dependent. The goal is to detect a column, or set (S) of columns, which might be best suited as the primary key for each table. For a given source database, the primary key analysis may result in identifying anywhere from none to one or
15 more candidate dependencies that are best suited as "candidate primary keys".

The primary key analysis uses the results obtained from the table analysis process 152 (Figure 4), which identifies the dependencies among the columns of a table, and one or more of these dependencies will potentially become candidate keys. The data repository entities used during the primary
20 key analysis procedure are the same as those shown in Figure 9 for the table analysis entities. The results of the primary key analysis are represented by the AggregateDependency entity 154 of Figure 9. A dependency which is found to be a candidate key is flagged in the corresponding instance (the column's associated source database table) of the AggregateDependency entity.

25 Following completion of the primary key analysis 162 (Figure 4), and a pause 164 to permit the operator to accept or modify the primary key inferences made by the invention, the process next performs a cross table analysis procedure 166. Cross table analysis 166 is the process of comparing each column in each source database table with each column in each other table of
30 the source database to detect columns that share a common "domain". A

domain is the set of data values found in a column. If columns from different tables share a common domain, it may indicate the existence of a relationship between the two tables, or, redundant data.

The entities of the data repository 37 (Figure 1) which are used during
5 the cross table analysis are shown in the summary entity diagram of Figure 10. The entities include the TableAnalysis entity 128 (Figure 7), which represents the results of each analyzed source table, and the ColumnAnalysisConfig entity 130 (Figure 7), which represents each analyzed source column. The cross table analysis procedure uses these entities to analyze the column domains of each
10 analyzed source table. A DomainsCompared entity 168 (with primary keys 169) is a child of the ColumnAnalysisConfig entity 130, and each instance of the DomainsCompared entity represents a pair of columns whose domains have been compared during cross table analysis. Any common domains which are found in the process are recorded in a CommonDomain entity 170 (with
15 primary keys 171).

A domain comparison is a bi-directional process. As an example, the process may conclude that a domain of column C1 of table T.1 (referred to here as column T1.C1) is contained to a substantial degree in column C2 of table T2 (i.e. column T2.C2). The process looks at the common domain from the
20 perspective of each column to make the distinction between dependency or redundant data. An instance of the CommonDomain entity 170 is created for each such conclusion. If no common domain is detected from the perspective of either column, then no instance of the CommonDomain entity is created.

The CommonDomain entity 170 is a child of the DomainsCompared
25 entity 168. Each instance of the CommonDomain entity 170 represents the fact that the domain of a "base" column T1.C1 is largely contained within the domain of a "paired" column T2.C2, where T2 is not the same table as T1. A common domain occurrence is determined from the perspective of the base column. It should be understood that, for a given pair of columns in the DomainsCompared
30 entity 168, there may be zero, one, or two rows in the CommonDomain entity

170. Also the degree to which the domain of the base column is contained within the domain of the paired column is indicated by a calculated percentage of domain intersection. A common domain is only noted if the percentage meets or exceeds the CommonDomainThreshold configuration setting in the DbAnalysis entity (see the DbAnalysis entity listing in Appendix I). Appendix I lists all of the attributes of each of the Figure 10 entities used in this cross table analysis.

Referring again to Figure 4, following completion of the cross table analysis procedure 166, and following a review interval 172, the present invention next performs a relationship analysis procedure 174. This relationship analysis uses the results of the preceding cross table analysis to make inferences regarding the existence of relationships between columns in the source database tables. The existence of a common domain between T1.C1 and T2.C2, detected during Cross Table Analysis, may be indicative of a relationship between the two tables T1 and T2.

As an example, referring to Table A below, if it is determined that set S1 of columns C1, C2 in table T1 and set S2 of columns C7, C8 in table T2 have

Table A

Table Name	Columns Sharing Domains
T1	S1: {C1, C2}
T2	S2: {C7, C8}

the same number of columns per set, and that column C1 shares a common domain with column C7 and column C2 shares a common domain with column C8. If the primary key of table T2 consists exactly of all the columns (C7, C8) in set S2, it is then likely that tables T1 and T2 are related, with T2 as the parent in the relationship and T1 as the child. Furthermore, a foreign key of table T1, which relates child table T1 to the parent table T2, consists of exactly the columns in the set S1 (columns C1 and C2). If the above criteria for identifying a primary key/foreign key pairing cannot be satisfied, then the existence of a common domain will be indicative of redundant data rather than a relationship.

The data repository entities used during the relationship analysis process are shown in the summary entity relationship diagram of Figure 11. Any inferred table key, whether its a primary key or a foreign key, is represented by the InferredKey entity 176 (with primary keys 177) If it is a primary key it is related to the instance of the AggregateDependency entity 154 from which it was derived. If it is a foreign key, it is related to an instance of the CommonDomain entity 170 from which it was derived.

Every column that makes up a key is noted by an occurrence of the InferredKeyColumnMap entity 178 (with primary keys 179), which relates the key column to the analyzed table column via the ColumnAnalysisConfig entity 130. Any inferred relationships between source database tables are represented by the InferredRelation entity 180 (with primary key 181). The primary key of the parent table, and the foreign key of the child table, are noted via parent/child relationships between InferredRelation entity 180 and an InferredKey entity 182 (with primary keys 183). The TableNameMap entity 184 (with primary keys 185) is used to map a given table to all related occurrences of the inferred relationships (InferredRelation entity 180) and the inferred keys (InferredKey entity 182). The full list of attributes of each of the identified entities in Figure 11 is shown in Appendix I under the title of the particular entity.

As shown in Figure 11, and as known to those skilled in the art of computer programming, the diamond-like symbols shown associated with the TableAnalysis entity 128, the ColumnAnalysisConfig entity 130, the AggregateDependency entity 154, the InferredRelation entity 180, and the Inferred Key entity 182, signify a “non-identifying relationship” between these “parent” entities and their related child entity. A non-identifying relationship is one in which the child does not get its primary key from the parent. The relationship with the parent is optional, and this is shown with the diamond symbol and the dashed relationship lines. NULLs in the columns of the child entity are permitted.

As shown in Figure 4, following completion of the relational analysis procedure 174, the method 78 provides for another pause 186 to permit the operator to review the inferences made in the relational analysis and to accept or modify them on an individual basis. At this time the source data analysis is
5 complete. In the best mode embodiment, however, the invention normalizes the source data in step 188. As may be known, normalization refers to the process of transforming a database schema into a form that by design prevents the occurrence of various update anomalies.

The data repository models data that is in at least 1st normal form (i.e. no
10 repeating groups of data items). The data repository supports processes that transform 1st or 2nd normal form data into 3rd normal form and beyond. A table in 3rd normal form can contain only columns that are fully dependent upon the primary key. Because the data repository encompasses complete metadata for a fully analyzed database, all the information required to automatically normalize
15 a table and database is available. The data repository entities used during normalization are shown in the summary entity relationship diagram of Figures 12A and 12B.

If an AggregateDependency entity 154 exists for a given table, and that dependency is not the basis for the primary key, then the table is not in 3rd
20 normal form. The process of normalization therefore considers all such extraneous dependencies as recorded using the Aggregate Dependency entity.

If an extraneous dependency D exists, then the original table A is split into two tables B and C; each of these is represented by a NewDbTable entity 190 (with primary keys 191). An OldTable NewTableMap entity 192 (with
25 primary keys 193) records the relationship between the original table A and the new tables B and C that are derived from table A.

All columns in the original table A that are not involved in the dependency D are placed in new table B, using a NewDbColumn entity 194 (with primary keys 195), which is a child of the NewDbTable entity 190. The
30 dependent columns of dependency D are placed in the new table C (using the

NewDbColumn entity 190) along with the determinant columns of dependency D. These dependent columns then form the primary key of new table C, which is recorded using the InferredKey entity, along with the InferredKeyColumnMap entity 178 to map the key columns back to the original analyzed columns.

The determinant columns of dependency D are also placed in the new table B, because they form the basis for the relationship that must exist between new tables B and C. These determinant columns in new table B are used to define a foreign key using the InferredKey entity 182, and the relationship between new tables B and C is recorded using the InferredRelation entity 180. The instance of a InferredRelation entity 180 links the InferredKey entity instance for the foreign key of new table B with the InferredKey instance for the primary key of new table C.

Occurrences of all newly created instances of the NewDbTable and NewDbColumn entities are related to the originating dependency D through use of relationships back to the AggregateDependency instance. As shown in Figure 4, following the normalization step 188 the method exits the process step 78 at 198.

The source metadata obtained by the present method and stored in the data repository, may be transformed into a wide variety of EDM applications. The architecture of the data repository includes a construct known as a migration specification. The migration specification is a complete and detailed map for producing one or more target tables, each from any number of source tables. The architecture supports generation of many types of ETL jobs using migration specifications. A. given specification is generic in nature, rather than assuming a specific ETL vendor technology. This generic nature is the key to the ability to automatically generate many different jobs from a single specification.

Figure 13, which is a summary entity relationship diagram, illustrates all of the entities and relationships used with the migration specifications. The full

list of attributes for each of the Figure 13 entities can be found in Appendix I under the related entity heading. The MigrationSpec entity 200 (with primary key 201) represents migration specifications. Specifications can include additional, customer properties as defined in the MigrationSpecPropertyList entity 202 (with primary key 203) and MigrationSpecProperty entity 204 (with primary key entity 205). A migration specification is focused upon the concept of defining the procedures required to produce the data values for a single column.

The process by which every target table column is produced is defined by the ColumnMapping entity 206 (with primary key 207). The transformation expression (TransformExpr- see Appendix I) is an attribute of the ColumnMapping entity 206. The transformation expression can include any number of source column references, along with additional specifications that transform data, including (but not limited to): invocations of functions; simple or complex arithmetic operators; and string manipulation.

Production of a target column may also include direct mappings of source column values using mapping tables that are either specific to the migration specification (the MapXLate entity 208, with primary keys 209) or are globally defined for use within more than one migration specification (the GlobalXLate entity 210, with primary keys 211).

The single target column to be produced via an instance of the ColumnMapping entity 206 is linked to the definition of that target column (within the appropriate table of the target database) via a relationship to the corresponding DictColumn entity 86 instance.

Every source column referenced in the transformation expression for a given target column is similarly linked to the definitions of the corresponding source columns (within the appropriate table of the source database) via a relationship to the corresponding DictColumn entity instance.

Not every source column will be referenced in a migration specification that references the source table containing that column. This can occur, for

instance, if a source database is being used to produce a data warehouse in which only a portion of the data will be available. If such a source column is to be deliberately excluded, that may be noted by creation of an instance of the SourceColumnExclusion entity 212 (with primary keys 213). This can be used
5 to omit warnings during validation of a migration specification.

Migration specifications are often used to produce target data using a process that must combine data from multiple source tables. In such cases, data from the source tables must be combined in an appropriate manner to yield correct results during the process of building the target data. The existing
10 relationships within the source data must therefore be utilized. Because these relationships are known within the data repository via the DictRelation entity 118, the architecture supports automatic identification of all possible sequences in which all required source tables might be linked, or joined.

Note that source tables may be directly referenced in a migration
15 specification (in the transformation expression), yet additional source tables often must be involved in the sequence of joining tables. This occurs when the referenced source tables are not all directly related to one another, and must be indirectly joined via other source tables.

Each possible sequence of joining source tables is noted as an instance
20 of the JoinPath entity 214 (with primary key 215). The joining of a pair of tables in the path is noted by an ordered instance of the JoinPath entity 214. Each such instance of the JoinPathStep entity 216 (with primary keys 217) instance is related to the single DictRelation instance that supports that table join.

25 The comprehensive nature of the source metadata obtained with the present method, and the manner in which this metadata is stored with the structural model of the data repository, provide a data engine for a wide variety of EDM applications. An example of one EDM application with which the present data repository is an OLAP (On-Line Analytical Processing) cube. As
30 known an OLAP cube is a multi-dimension structure often used for presenting

data for enhanced analysis. It generally presents data items as either dimensions or measures.

The data repository includes the ability to model OLAP cubes in a generic manner that permits, in conjunction with other Repository information, the automatic generation of cube definitions and/or cube construction jobs using a variety of vendor-specific protocols. Figure 14 is a summary entity relationship diagram illustrating all the entities involved with modeling OLAP cubes, together with their primary keys. The attributes of these entities are listed in Appendix I under their respective entity heading.

Referring to Figure 14, each unique cube definition is represented by the OLAPCube entity 218 (with primary key 219). The source tables referenced in building the cube are noted by the CubeSourceTableMap entity 220 (with primary keys 221), each occurrence of which is related to the instance of the DictTable entity 85 that defines a referenced source table. As with the JoinPath and JoinPathStep entities (214, 216, Figure 13) used with migration specifications, the CubeJoinPath entity 222 (with primary keys 223) and the CubeJoinPathStep entity 224 (with primary keys 225) serve to define the possible sequences of source table joins needed to produce the cube data.

Each cube dimension is defined using the CubeDimension entity 226 (with primary keys 227). Each instance of a dimension is related back to the instance of the DictColumn entity 86 from which it is formed. A cube can be defined to include dimension hierarchies, which are modeled using the self-referencing relationship on the CubeDimension entity 226. That is, a named dimension may consist of a group of other named dimensions, any of which in turn may consist of other dimensions. There is no limit to the number of levels that may be modeled as a dimension hierarchy.

Each cube measure is defined using the CubeMeasure entity 228 (with primary keys 229). As with dimensions, each instance of a measure is related back to the DictColumn instance from which it is formed.

The variety of DM applications which may be generated from the metadata repository of the present invention includes, but is not limited to:

- (i) Materials in a form suitable for presentation, such as web pages, OLAP cubes, or spreadsheets;
- 5 (ii) Computer software that can be used to produce data and/or presentation materials;
- (iii) Instructions for external software packages, which are invoked to transform data or produce presentation materials;
- (iv) Data models, which provide the definition of data sources such as
- 10 relational databases;
- (v) A specification describing how to migrate (transform) data from one or more sources to a target;
- (vi) Data in a desired target form from one or more sources; and
- (vii) identifying the similarities among the many types of desired
- 15 information.

The architecture provided by the present invention provides benefits to the user in terms of increased accuracy, and reduced time and cost, to build EDM applications. It also benefits manufactures in allowing for rapid development of processes that generate new types of materials. Development cost and therefore

20 time-to-market is dramatically reduced.

The metadata-based repository is a key component of the architecture. "Metadata-based" means that the structure of the repository is metadata-centric; that is, the atomic elements of the structure describe the characteristics of the external data that is being modeled. Many areas of information are addressed

25 by the characteristics of the data, including:

- (i) The data type characteristics of individual data elements;
- (ii) The grouping of elementary data items into a collection, such as a file or table;
- (iii) The grouping of collections into an aggregate database;
- 30 (iv) The relationships between data elements within collection;

- (v) The relationships between collections of data elements; and
- (vi) The relationships between data elements of different collections.

Although the invention has been shown and described with respect to a best mode embodiment thereof, it should be understood by those skilled in the art that various changes, omissions, and additions may be made to the form and detail of the disclosed embodiment without departing from the spirit and scope of the invention, as recited in the following claims.

5